

ModernMasterMind									
	Code				Hints				
01	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
02	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
03	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
04	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
05	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
06	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
07	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
08	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
09	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
10	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
11	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
12	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	

Groen:1 Pin correct  
Geel:1 Kleur komt voor

# ModernMasterMind

PROJECT I – ICT-ELEKTRONICA

## 1 Voorwoord

Ik ben Dries Kennes, student ICT-Elektronica Fase 1 aan Thomas More Mechelen op Campus De Nayer. Mijn voornaamste interesses zijn elektronica en alles wat te enigszins programmeren valt. Online ga ik door het leven als Driesoo7, een enthousiaste Java programmeur die zich in zijn vrije tijd vooral bezighoudt met het maken van build-to-order Minecraft mods.

Tijdens de kerstvakantie ben ik op zoek gegaan naar projectonderwerpen. Vorig jaar (6e jaar middelbaar) wilden wij tijdens de lessen elektronica met een Arduino en een LED matrix spelletjes maken. Helaas was het juiste materiaal niet tijdig beschikbaar. Daarom heb ik er dit jaar voor gekozen om één van die spellen uit te werken. Het werd Mastermind omdat dit een uitdagende leek, onder andere dankzij de vereiste om kleuren te kunnen weergeven.

Mijn contacteren kan via email, [dries.kennes@student.thomasmore.be](mailto:dries.kennes@student.thomasmore.be), of via mijn website, [driesoo7.net](http://driesoo7.net).

Ten slotte wil ik Marc Roggemans bedanken voor het uitlenen van materiaal en Jurre De Weerdts voor het helpen met solderen van enkele SMD onderdelen.

## 2 Inhoud

1	Voorwoord .....	1
2	Inhoud .....	2
3	Hardware.....	3
3.1	RIOT.....	3
3.2	Het basisprincipe.....	3
3.3	De PCBs.....	4
4	Software.....	5
4.1	De memory map .....	5
4.2	Het WS2812 protocol .....	6
5	Besluit.....	8
6	Bijlagen.....	9
6.1	PCB Schema's & Layout.....	9
6.2	Broncode .....	12
6.2.1	mastermind.h.....	12
6.2.2	mastermind.c .....	15
6.2.3	HTML template .....	31
6.2.4	AVR.h .....	33
6.2.5	AVR.c.....	37

## 3 Hardware

### 3.1 RIOT

RIOT staat voor Remote Internet Operated Terminal, en is een ontwerp van M. Roggemans en D. Pauwels. RIOT werd ontworpen als educatief platform in 2002, maar is eigenlijk nooit gebruikt. Meneer Roggemans heeft tijdens de lessen interfacetechnieken dit platform wel aangehaald, en zo kreeg ik interesse om dit te gebruiken. Ik kreeg toegang tot de bordjes na de examens na de kerstvakantie, zodat ik hiermee kon experimenteren, en zien of ze bruikbaar zouden zijn als basis voor mijn project. Het gebruiken van een 10+ jaar oud platform zorgt voor een aantal hindernissen, meer hierover in het hoofdstuk *Software*.

Meer informatie over RIOT is beschikbaar in op Telescript<sup>1</sup>.

### 3.2 Het basisprincipe

Het basisprincipe van RIOT (en dus ook mijn project) is dat 2 controllers met elkaar communiceren via een gedeeld geheugen (een Dual Port RAM, voortaan DP-RAM). Dit geheugen heeft een aantal 'gewone' adressen die vrij kunnen worden gebruikt, en 2 interrupt adressen. De interrupt adressen worden gebruikt om, zoals de naam al doet raden, van de ene controller naar de andere een interrupt te genereren. De controllers zijn een Beck SC12 en een AVR ATMeg128A.

De reden voor het gebruiken van een dergelijk systeem is in dit geval de hoeveelheid IO verhogen. De SC12 heeft namelijk een zeer beperkte hoeveelheid IO, die vooral op pinnen zit die al door andere functies in gebruikt zijn (bijvoorbeeld UART of I<sup>2</sup>C). De SC12 heeft echter wel een gemultiplexte 8 bit data/adres bus, waarmee een extern geheugen van 256 bytes kan worden aangesproken. Door 2 IO pinnen te gebruiken als adres lijnen 8 en 9 kan het aanspreekbare geheugen worden uitgebreid tot 1kb (4 pages van 256 bytes).

Aangezien een Mastermind spelbord 12 rijen heeft van 8 gekleurde (RGB) pionnen, dit zijn in totaal dus  $12 \times 8 \times 3 = 288$  bytes is dit trucje geen overbodige luxe. Als men dan nog op een eenvoudige manier een LCD wil aansturen (20 karakters x 4 lijnen) heeft men nog eens 80 bytes nodig. Uiteindelijk zou 512 bytes genoeg zijn geweest, maar een DP-RAM chip met 1kb is commercieel beschikbaar, 512 bytes niet.

Aangezien de adres- en databus van beide controllers gemultiplext zijn, is het nodig om een adres latch te gebruiken. Ik gebruik 74AHC573 chips aangezien die aan de timingspecificaties van de ATMeg128A voldoen.

Voor de LEDs gebruik ik WS2812 LEDs. Deze LEDs zijn ideaal voor dit project omdat ze, ongeacht de hoeveelheid LEDs, maar 1 pin op een (relatief snelle) controller nodig hebben. Ze worden namelijk allemaal in serie geschakeld. De werking van het protocol word uitgelegd in het hoofdstuk '*Het WS2812 protocol*'.

---

<sup>1</sup> <ftp://193.191.150.44/pub/CD-Microcontrollers/RIOT/>

### 3.3 De PCBs

Op mijn versie van het bord zijn onnodige onderdelen weggelaten. Onder andere de UART naar RS232, de Real Time Clock en het voedingscircuit moesten er aan geloven.

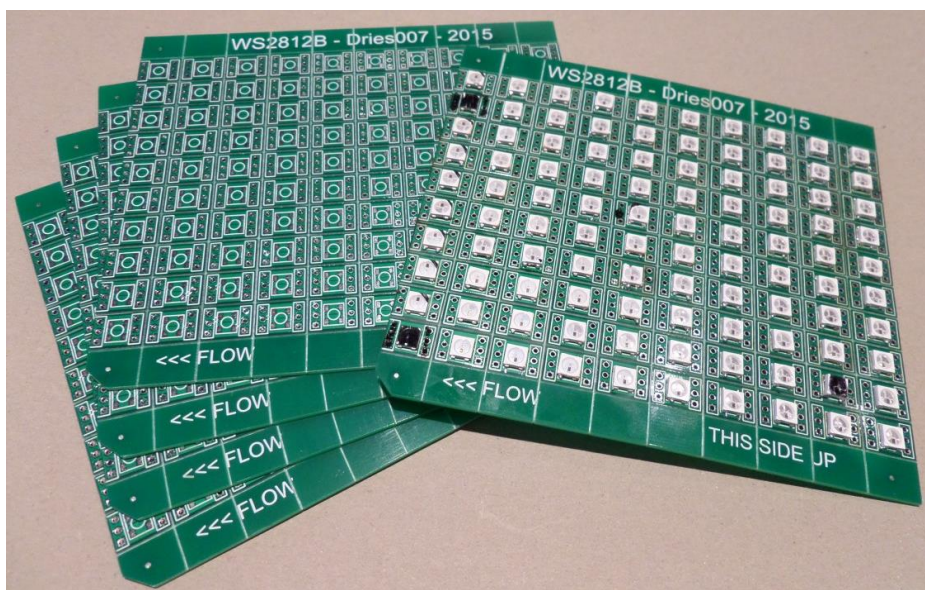


*Figuur 3.1 RIOT links, Eigen PCB rechts*

De PCBs zijn getekend met Altium. Het main bord en de LCD breakout zijn besteld bij Seedstudio, het LED panel bij multi-cb. De bestukking was manueel. De volgende fouten heb ik achteraf (tijdens het testen of solderen) ontdekt:

- De databus van het DP-RAM naar de AVR is in omgekeerde volgorde (de bits zitten omgekeerd).
- De WS2812 LEDs passen maar net op de getekende footprint, wat wil zeggen dat ze bijna onmogelijk met een gewone soldeerbout soldeerbaar zijn. Gelukkig heb ik op school hulp gekregen, en mocht ik een warme lucht soldeerstation gebruiken.

Het volledige schema en de layout van de PCBs is beschikbaar in de bijlage *PCB Schema's & Layout*.



*Figuur 3.2 Het LED panel*

## 4 Software

### 4.1 De memory map

Dit is de verdeling van het DP-RAM. De 2 interrupt adressen liggen vast, namelijk 0x3FE en 0x3FF.

Adres	Functie
0x000 -> 0x1FE	RGB data voor maximaal 170 LEDs
0x1FF	Aantal LEDs
0x200 -> 0x250	LCD buffer (80 karakters)
0x251	LCD commando
0x252	LEDs dim
0x253	Laatst ingedrukte toets op keypad
0x245 -> 0x3FD	Vrij geheugen
0x3FE	Interrupt SC12 -> AVR
0x3FF	Interrupt AVR -> SC12

Tabel 4.1 De geheugenallocatie van het DP-RAM

Het interrupt adres 0x3FE word gebruikt als een commando register van SC12 -> AVR.

Waarde	Functie
0x00	Geen functie
0x01	Update LEDs
0x02	Print LCD karakter buffer (tot max of 0x00)
0x03	Stuur LCD instructie
0x04	Stuur clear instructie en print LCD karakter buffer
0x05	Zet LCD cursor positie op LCD commando
0x06	LCD backlight aan
0x07	LCD backlight uit

Tabel 4.2 De commando's van de SC12 naar de AVR

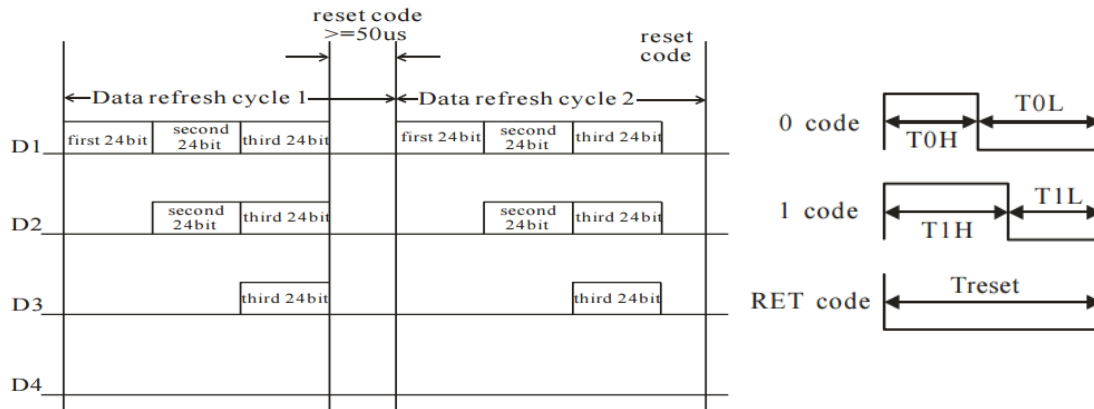
Het interrupt adres 0x3FF word gebruikt als een commando register van AVR -> SC12.

Waarde	Functie
0x00	Geen functie
0x01	Keypress

Tabel 4.3 De commando's van de SC12 naar de AVR

## 4.2 Het WS2812 protocol

De WS2812 LEDs werken als een lange serie schakeling, waarbij elke led de eerste 24 bits gebruikt om zijn kleur in te stellen. De andere bits worden doorgegeven, zie *Figuur 4.1*. De volgorde van de bits is niet RGB, maar GRB, met hoogste bit eerst.



Figuur 4.1 LED cascading & timing diagram

Code	Betekenis	Tijd	Min	Typ	Max
T0H	0 code - high voltage time	0,35 µs ±150 ns	200 ns	350 ns	500 ns
T1H	1 code - high voltage time	0,70 µs ±150 ns	550 ns	700 ns	850 ns
T0L	0 code - low voltage time	0,80 µs ±150 ns	650 ns	800 ns	950 ns
T1L	1 code - low voltage time	0,60 µs ±150 ns	450 ns	600 ns	750 ns
Reset	low voltage time	> 50µs	50 µs		
TH+TL	Cyclus time	1,25µs ±600ns	660 ns	1250 ns	1850 ns

Tabel 4.4 De timing tabel uit de datasheet, met genormaliseerde waarden

De software driver die vaak voor de ze LEDs word gebruikt (Adafruit NeoPixel Library<sup>2</sup>) is ingewikkeld, geschreven in C++ en geschikt voor zowel 8, 12 als 16 MHz Arduino's. Na verder zoeken vond ik een aantal interessante artikels<sup>3</sup> die de timing van de LEDs analyseren.

Uit hun analyse blijkt dat de timing eigenlijk niet zo restrictief is als lijkt uit de datasheet. De enige kritieke tijd blijkt T0H te zijn, en die is net haalbaar op 4MHz (500ns \* 4MHz = 2 klokcycli). Alle andere timing restricties zijn langer, en kunnen dus met nops worden ingevuld.

Uiteindelijk heb ik ervoor gekozen een aangepaste versie van de *light\_ws2812 library*<sup>4</sup> te gebruiken. Dit is de code die word beschreven in een van de artikels. Ze is simpel en eenvoudig te begrijpen.

<sup>2</sup> [github.com/adafruit/Adafruit\\_NeoPixel](https://github.com/adafruit/Adafruit_NeoPixel)

<sup>3</sup> [wp.josh.com/2014/05/13/ws2812](http://wp.josh.com/2014/05/13/ws2812) en [cpldcpu.wordpress.com/2014/01/14/light\\_ws2812-](http://cpldcpu.wordpress.com/2014/01/14/light_ws2812-)

<sup>4</sup> [github.com/cpldcpu/light\\_ws2812/tree/master/light\\_ws2812\\_AVR/Light\\_WS2812](https://github.com/cpldcpu/light_ws2812/tree/master/light_ws2812_AVR/Light_WS2812)

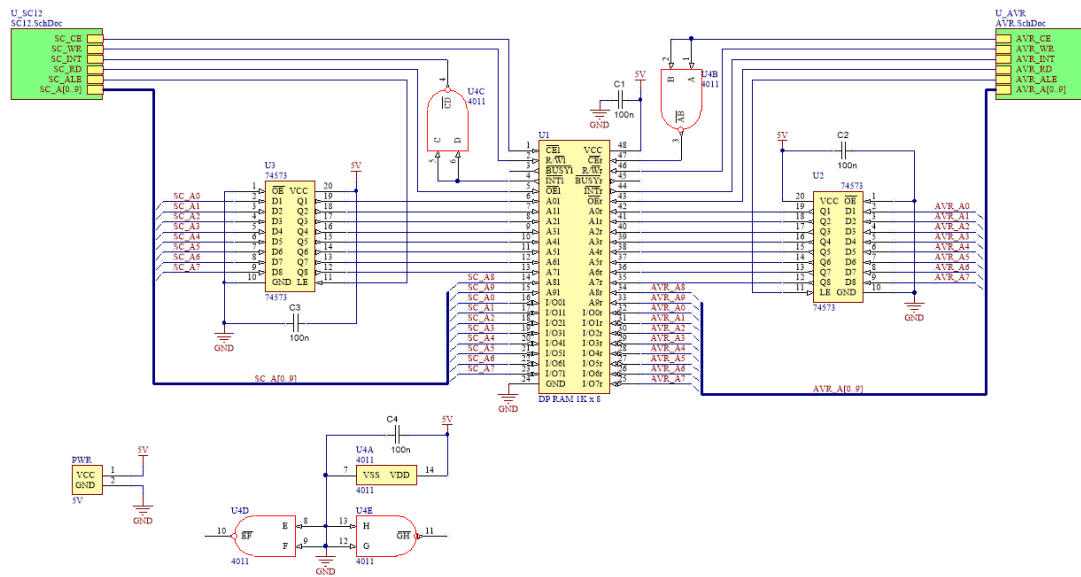
## 4.3 De AVR software



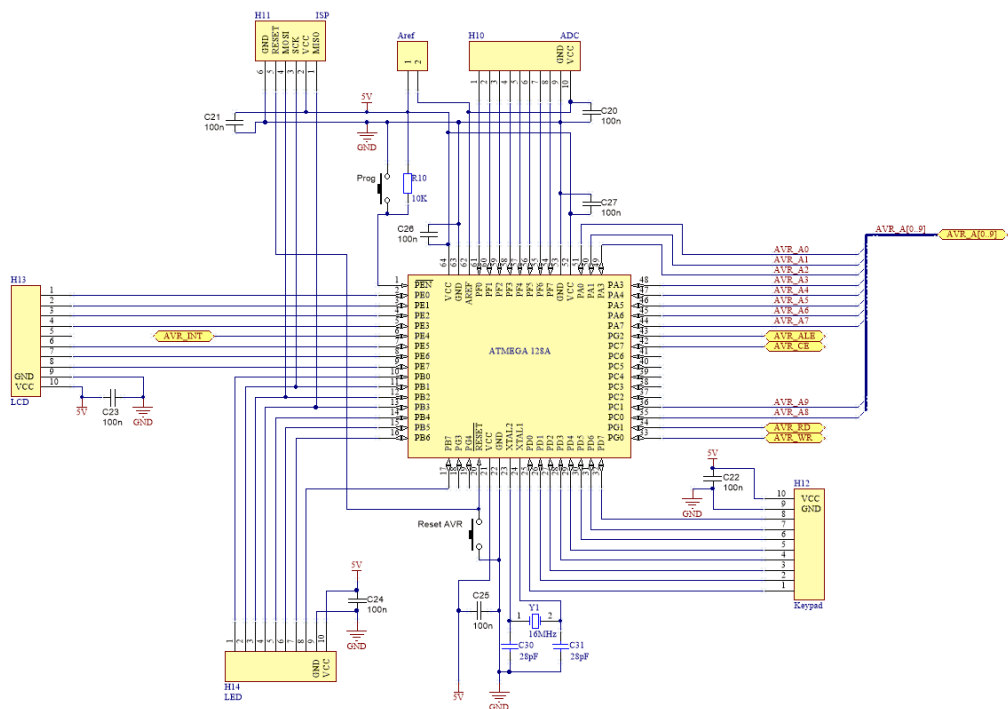
## 5 Besluit

## 6 Bijlagen

### 6.1 PCB Schema's & Layout

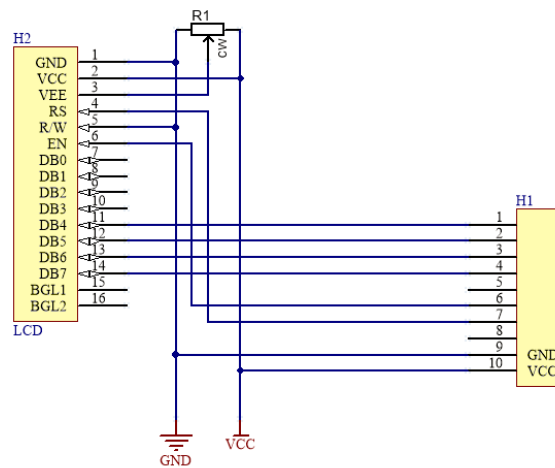


Figuur 6.1 Schema DP-RAM deel

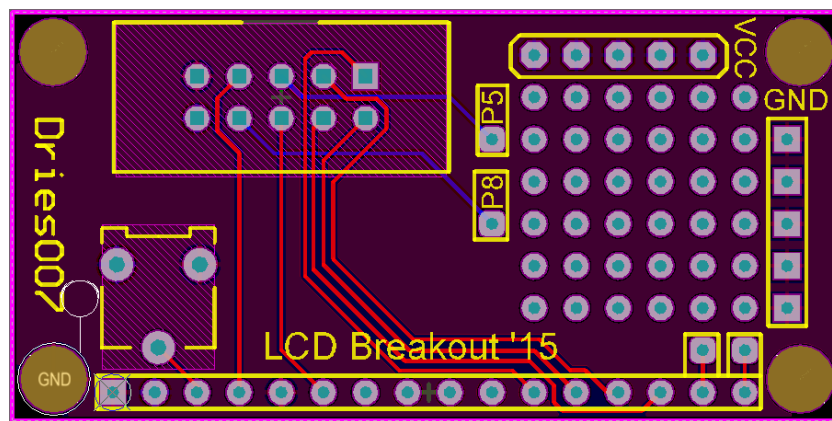


Figuur 6.2 Schema AVR deel

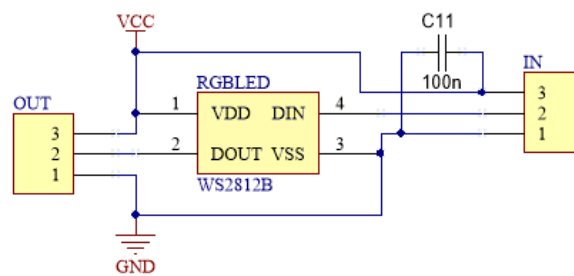




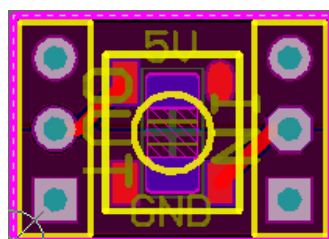
Figuur 6.5 Schema LCD-Breakout



Figuur 6.6 LCD-Breakout PCB layout (50 x 25 mm)



Figuur 6.7 WS2812B Schema



Figuur 6.8 WS2812B Layout (11 x 8 mm)

## 6.2 Broncode

De volledige broncode is ook beschikbaar op [github.com/driesoo7/ModernMasterMind](https://github.com/driesoo7/ModernMasterMind).

Mastermind.c draait op de SC12, AVR.c op de ATMegai28.

Het originele HTML template is ook toegevoegd, aangezien dit gecomprimeerd bijna onleesbaar is.

De cLib library die bij mastermind.h code hoort is beschikbaar op [www.beck-ipc.com](http://www.beck-ipc.com).

### 6.2.1 mastermind.h

```
1  #ifndef SRC_MASTERMIND_H_
2  #define SRC_MASTERMIND_H_
3
4  #pragma option -1 //create 80186 code
5
6  #include <clib.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include <limits.h>
11 #include <dos.h>
12 #include <stdarg.h>
13 #include <ctype.h>
14 #include <rtos.h>
15 #include <i86.h>
16
17 #include "ramdump.h"
18 #include "httpcli.h"
19 #include "base64.h"
20 #include "dns.h"
21
22 /* ===== DEFINES ===== */
23
24 #define DEBUG 0
25
26 #define VERSION "0.2"
27 #define CYEAR "15"
28
29 #define TASK_STACKSIZE 2048
30
31 #define TCPIP_int 0xAC
32
33 #define LCD_LINE_SIZE 16
34 #define LCD_LINES 2
35
36 #define MAX_COLORS 8
37 #define COLORS 4
38 #define ROWS 12
39
40 #define STATE_NO_GAME 0
41 #define STATE_GAME_CONFIGURED 1
42 #define STATE_GAME_STARTED 2
43 #define STATE_GAME_OVER 3
44 #define STATE_GAME_WON 4
45
46 // amount of (positive) hours offset from GMT
47 #define TIMEZONE_OFFSET 2
48
49 // 170 Leds max (x 3 bytes = 0x1FE)
50 #define RAM_LEDS_START 0x000
51 #define RAM_LEDS_END 0x1FE
52 #define RAM_LEDS_AMOUNT 0x1FF
```

```

53 // 80 bytes of char buffer for LCD
54 #define RAM_LCD_START    0x200
55 #define RAM_LCD_END      0x250
56 #define RAM_LCD_CMD      0x251
57 // Mask for global LED dimming
58 #define RAM_LEDS_DIM      0x252
59 #define RAM_KP_LASTKEY    0x253
60
61 #define RAM_VERSION_1     0x300
62 #define RAM_VERSION_2     0x301
63
64 // Interrupt registers
65 #define RAM_INT_SEND      0x3FF
66 #define RAM_INT_GET       0x3FE
67
68 #define MAX_LEDS           (RAM_LEDS_END - RAM_LEDS_START)
69 #define MAX_LCD_CHARS     (RAM_LCD_END - RAM_LCD_START)
70
71 #define CMD_LEDS_SEND      0x01
72 #define CMD_LCD_CHAR       0x02
73 #define CMD_LCD_CMD        0x03
74 #define CMD_LCD_CL_PR     0x04
75 #define CMD_LCD_POS        0x05
76 #define CMD_LCD_BL_ON     0x06
77 #define CMD_LCD_BL_OFF    0x07
78
79 #define SATUS_KP_PRESS    0x01
80
81 #define NORMALIZE_ADDRESS(addr) ((addr & 0x0FF) | 0x100)
82 #define BANK_FROM_ADDRESS(addr) (addr >> 8)
83
84 /* ===== STRUCTS ===== */
85
86 typedef unsigned char byte;
87 typedef unsigned short address;
88
89 typedef struct
90 {
91     unsigned long ip;
92     char name[21];
93 } User;
94
95 typedef struct
96 {
97     byte r;
98     byte g;
99     byte b;
100 } RGB;
101
102 typedef struct
103 {
104     byte state;
105     byte vsPlayer;
106     User * host;
107     byte colors;
108     byte code[COLORS];
109     byte nrOfGuesses;
110     byte guesses[ROWS][COLORS + 2];
111 } Game;
112
113 struct userlist_el
114 {
115     User user;
116     struct userlist_el * next;
117 };
118
119 /* ===== METHODS ===== */

```

```

120
121 void endProgram();
122
123 User * getUserByIP(long far * ip);
124 User * getUserByName(char * name);
125 void addUser(long ip, char name[21]);
126
127 Game * getGame();
128 void resetGame();
129 void setRndCode(byte colors);
130 void guessRow(byte id);
131
132 void enableDatabus(); // Enables databus
133 byte readDatabus(address addr); // read byte from databus
134 void writeDatabus(address addr, byte value); // write byte to databus
135 //void initTime();
136
137 void clearLCD();
138 void setLCDLine(byte line, const char *string);
139 void setLCDLineFormat(byte line, const char *format, ...);
140
141 void installCGIMethods();
142 void removeCGIMethods();
143
144 void printAllUsers();
145
146 /* ===== GLOBALS ===== */
147
148 const RGB BLACK = { 0, 0, 0 };
149 const RGB WHITE = { 255, 255, 255 };
150
151 const RGB RED = { 255, 0, 0 };
152 const RGB GREEN = { 0, 255, 0 };
153 const RGB BLUE = { 0, 0, 255 };
154
155 const RGB PINK = { 255, 0, 255 };
156 const RGB AQUA = { 0, 255, 255 };
157 const RGB YELLOW = { 255, 255, 0 };
158
159 const RGB PURPLE = { 130, 0, 255 };
160 const RGB ORANGE = { 255, 130, 0 };
161
162 const RGB ALL_COLORS[10] = { { 255, 0, 0 }, { 0, 255, 0 }, { 0, 0, 255 }, {
163     255, 0, 255 }, { 0, 255, 200 }, { 255, 255, 0 }, { 255, 255, 255 }, {
164     255, 80, 0 }, { 0, 0, 0 } };
165 const char * ALL_COLOR_CLASSES[10] = { "red", "green", "blue", "pink",
166     "aqua", "yellow", "white", "orange", "black" };
167
168 union REGS inregs;
169 union REGS outregs;
170 struct SREGS segregs;
171
172 Game game;
173
174 struct userlist_el * listHead = NULL;
175 struct userlist_el * listTail = NULL;
176
177 #endif /* SRC_MASTERMIND_H_ */

```

### 6.2.2 mastermind.c

```
1  #include "mastermind.h"
2
3  /*****
4   *                               USER RELATED
5   *****/
6
7  User * getUserByIP(long far * ip)
8  {
9      struct userlist_el * current = listHead;
10     while (current != NULL)
11     {
12         if (current->user.ip == *((unsigned long *) ip))
13             return &(current->user);
14         current = current->next;
15     }
16     return NULL;
17 }
18
19 User * getUserByName(char * name)
20 {
21     for (int i = 0; name[i]; i++)
22         name[i] = tolower(name[i]);
23
24     struct userlist_el * current = listHead;
25     while (current != NULL)
26     {
27         if (strcmp(current->user.name, name) == 0) return &(current->user);
28         current = current->next;
29     }
30     return NULL;
31 }
32
33 void addUser(long ip, char name[21])
34 {
35     for (int i = 0; name[i]; i++)
36         name[i] = tolower(name[i]);
37     struct userlist_el * newItem = (struct userlist_el *) malloc(
38         sizeof(struct userlist_el));
39
40     newItem->user.ip = ip;
41     strcpy(newItem->user.name, name);
42
43     if (listHead == NULL)
44     {
45         listHead = listTail = newItem;
46     }
47     else
48     {
49         listTail->next = newItem;
50     }
51 }
52
53 void printAllUsers()
54 {
55     if (listHead == NULL)
56     {
57         printf("No users in the user list.\n");
58         return;
59     }
60
61     printf("User list:\n");
62     struct userlist_el * current = listHead;
63     while (current != NULL)
64     {
65         printf("Username: %s Remote IP: %d.%d.%d.%d\n", current->user.name,
66             (int) ((current->user.ip & 0xFF000000) >> 24),
```



```

67         (int) ((current->user.ip & 0x00FF0000) >> 16),
68         (int) ((current->user.ip & 0x0000FF00) >> 8),
69         (int) (current->user.ip & 0x000000FF));
70     current = current->next;
71 }
72 }
73
74 /*****
75  *           LEDS RELATED
76  *****/
77
78 address sendProper(byte row, address addr)
79 {
80     for (byte p = 0; p < COLORS; p++)
81     {
82         RGB rgb = ALL_COLORS[game.guesses[row][p]];
83         writeDatabus(addr++, rgb.r);
84         writeDatabus(addr++, rgb.g);
85         writeDatabus(addr++, rgb.b);
86     }
87
88     byte g = game.guesses[row][COLORS];
89     byte r = game.guesses[row][COLORS + 1];
90
91     for (byte p = 0; p < g; p++)
92     {
93         writeDatabus(addr++, GREEN.r);
94         writeDatabus(addr++, GREEN.g);
95         writeDatabus(addr++, GREEN.b);
96     }
97
98     for (byte p = 0; p < r; p++)
99     {
100         writeDatabus(addr++, YELLOW.r);
101         writeDatabus(addr++, YELLOW.g);
102         writeDatabus(addr++, YELLOW.b);
103     }
104
105     for (byte p = g + r; p < COLORS; p++)
106     {
107         writeDatabus(addr++, BLACK.r);
108         writeDatabus(addr++, BLACK.g);
109         writeDatabus(addr++, BLACK.b);
110     }
111
112     return addr;
113 }
114
115 address sendReverse(byte row, address addr)
116 {
117     byte g = game.guesses[row][COLORS];
118     byte r = game.guesses[row][COLORS + 1];
119
120     for (byte p = g + r; p < COLORS; p++)
121     {
122         writeDatabus(addr++, BLACK.r);
123         writeDatabus(addr++, BLACK.g);
124         writeDatabus(addr++, BLACK.b);
125     }
126
127     for (byte p = 0; p < r; p++)
128     {
129         writeDatabus(addr++, YELLOW.r);
130         writeDatabus(addr++, YELLOW.g);
131         writeDatabus(addr++, YELLOW.b);
132     }
133 }

```

```

134     for (byte p = 0; p < g; p++)
135     {
136         writeDatabus(addr++, GREEN.r);
137         writeDatabus(addr++, GREEN.g);
138         writeDatabus(addr++, GREEN.b);
139     }
140
141     for (byte p = 0; p < COLORS; p++)
142     {
143         RGB rgb = ALL_COLORS[game.guesses[row][COLORS - 1 - p]];
144         writeDatabus(addr++, rgb.r);
145         writeDatabus(addr++, rgb.g);
146         writeDatabus(addr++, rgb.b);
147     }
148
149     return addr;
150 }
151
152 void sendLEDS()
153 {
154     address addr = RAM_LEDS_START;
155     byte row = 0;
156     while (row < ROWS)
157     {
158         addr = sendProper(row++, addr);
159         addr = sendReverse(row++, addr);
160     }
161     writeDatabus(RAM_INT_SEND, CMD_LEDS_SEND);
162 }
163
164 /*****
165  *                               GAME RELATED
166  *****/
167
168 Game * getGame()
169 {
170     return &game;
171 }
172
173 void resetGame()
174 {
175     game.state = STATE_NO_GAME;
176     game.nrOfGuesses = 0;
177     for (int i = 0; i < ROWS; i++)
178     {
179         for (int j = 0; j < COLORS; j++)
180         {
181             game.guesses[i][j] = 9; // 9 = black
182         }
183         game.guesses[i][COLORS] = 0;
184         game.guesses[i][COLORS + 1] = 0;
185     }
186     sendLEDS();
187 }
188
189 void setRndCode(byte colors)
190 {
191     if (colors > MAX_COLORS) colors = MAX_COLORS;
192     for (byte i = 0; i < COLORS; i++)
193     {
194         game.code[i] = rand() % colors;
195     }
196 }
197
198 void guessRow(byte id)
199 {
200     if (id >= ROWS)

```

```

201     {
202         game.state = STATE_GAME_OVER;
203         return;
204     }
205     byte usedUpPins[COLORS];
206     for (byte i = 0; i < COLORS; i++)
207         usedUpPins[i] = 0;
208
209     for (byte i = 0; i < COLORS; i++)
210     {
211         // Exact matches
212         if (game.guesses[id][i] == game.code[i])
213         {
214             if (usedUpPins[i] == 1)
215             {
216                 game.guesses[id][COLORS + 1]--;
217             }
218
219             usedUpPins[i] = 1;
220             game.guesses[id][COLORS]++;
221             if (game.guesses[id][COLORS] == COLORS)
222             {
223                 game.state = STATE_GAME_WON;
224             }
225         }
226         else
227         {
228             for (byte j = 0; j < COLORS; j++)
229             {
230                 if (game.guesses[id][i] == game.code[j] && usedUpPins[j] == 0)
231                 {
232                     usedUpPins[j] = 1;
233                     game.guesses[id][COLORS + 1]++;
234                     break;
235                 }
236             }
237         }
238     }
239     sendLEDS();
240 }
241
242 /*****
243  *                               DATA BUS RELATED
244  *****/
245
246 void enableDatabus()
247 {
248     pfe_enable_bus(0xFF, 1); // all 8 bits enabled with ALE
249     pfe_enable_pcs(1); // Chip select 1 (PI04)
250     pfe_enable_pio(2, 5); // PI02 = output, low
251     pfe_enable_pio(3, 5); // PI03 = output, low
252 }
253
254 byte readDatabus(address addr)
255 {
256     byte bank = addr >> 8;
257     pfe_enable_pio(2, bank & 0x01 ? 4 : 5); // if bank is 1 or 3, set PI02
258     high, otherwise set PI02 low
259     pfe_enable_pio(3, bank & 0x02 ? 4 : 5); // if bank is 3 or 4, set PI03
260     high, otherwise set PI03 low
261     return hal_read_bus((addr & 0x0FF) | 0x100, 0xFFFF, 0x0000); // Read data
262     bus on corrected address (always in 0x100..0x1FF range)
263 }
264
265 void writeDatabus(address addr, byte val)
266 {
267     byte bank = addr >> 8;

```

```

268 // if bank is 1 or 3, set PIO2 high, otherwise set PIO2 low
269 pfe_enable_pio(2, bank & 0x01 ? 4 : 5);
270 // if bank is 3 or 4, set PIO3 high, otherwise set PIO3 low
271 pfe_enable_pio(3, bank & 0x02 ? 4 : 5);
272 // Write to data bus on corrected address (always in 0x100..0x1FF range)
273 hal_write_bus((addr & 0xFF) | 0x100, val, 0xFFFF, 0x0000);
274 }
275
276 /*****
277 * LCD RELATED
278 *****/
279
280 void clearLCD()
281 {
282     for (address addr = RAM_LCD_START; addr <= RAM_LCD_END; addr++)
283     {
284         writeDatabus(addr, ' ');
285     }
286     writeDatabus(RAM_LCD_CMD, 0x01);
287     writeDatabus(RAM_INT_SEND, CMD_LCD_CMD);
288 }
289
290 void setLCDLine(byte line, const char * text)
291 {
292     address offset = RAM_LCD_START;
293
294     #if LCD_LINES == 2
295         if (line == 1 || line == 3) offset += 40;
296     #elif LCD_LINES == 4
297         if (line % 2 != 0) offset += 40;
298         if (line > 1) offset += 0x14;
299     #endif
300
301     address i = 0;
302     for (; i < strlen(text); i++) // chars
303     {
304         writeDatabus(i + offset, text[i]);
305     }
306     for (; i < LCD_LINE_SIZE; i++) // spaces
307     {
308         writeDatabus(i + offset, ' ');
309     }
310     writeDatabus(RAM_INT_SEND, CMD_LCD_CL_PR);
311     delay(100);
312 }
313
314 void setLCDLineFormat(byte line, const char * format, ...)
315 {
316     byte buffer[LCD_LINE_SIZE + 1];
317     buffer[LCD_LINE_SIZE] = 0x00;
318
319     /* Start magic */
320     va_list aptr;
321     va_start(aptr, format);
322     vsprintf(buffer, format, aptr);
323     va_end(aptr);
324     /* End magic */
325
326     setLCDLine(line, buffer);
327 }
328
329 /*****
330 * TASKS
331 *****/
332
333 byte LCDupdateRunning;
334

```

```

335 void LCDUpdate()
336 {
337     LCDUpdateRunning = 1;
338     byte ip[16], oldIp[16];
339
340     while (LCDUpdateRunning)
341     {
342         Get_IPConfig(ip, NULL, NULL);
343
344         if (strcmp(ip, oldIp) != 0)
345         {
346             strcpy(oldIp, ip);
347             setLCDLine(1, ip);
348         }
349         RTX_Sleep_Time(2500);
350     }
351 }
352
353 unsigned int LCDUpdate_stack[TASK_STACKSIZE / sizeof(unsigned int)];
354 int LCDUpdateID;
355
356 TaskDefBlock LCDUpdateTaskDefBlock = { LCDUpdate, { 'L', 'C', 'D', ' ' },
357     &LCDUpdate_stack[TASK_STACKSIZE / sizeof(unsigned int)], // top of stack
358     TASK_STACKSIZE, // size of stack
359     0, // attributes, not supported
360     100, // lower priority than any system tasks
361     0, // time slice (if any), not supported
362     0, 0, 0, 0 // mailboxes
363 };
364
365 /*****
366  *                               MAIN
367  *****/
368
369 /*
370  * Ends all tasks, returns focus and exits
371  */
372 void endProgram()
373 {
374     setLCDLine(0, "Shutdown issued");
375
376     removeCGIMethods();
377
378     printf("Gracefully ending all tasks...\n");
379
380     LCDUpdateRunning = 0; // tell tasks to stop
381
382     RTX_Sleep_Time(3000); // give tasks time to end
383
384     printf("Killing the non obedient tasks...\n");
385     RTX_Delete_Task(LCDUpdateID);
386
387     printf("\nEND OF PROGRAM\n");
388     // Release input/output
389     BIOS_Set_Focus(FOCUS_BOTH);
390     exit(0);
391 }
392
393 void test()
394 {
395     writeDatabus(RAM_LEDS_DIM, 0xFF);
396     writeDatabus(RAM_LEDS_AMOUNT, 100);
397
398     address addr = RAM_LEDS_START;
399     for (byte r = 0; r < ROWS; r++)
400     {
401         for (byte p = 0; p < COLORS * 2; p++)

```

```

402     {
403         RGB rgb = ALL_COLORS[rand() % MAX_COLORS];
404         writeDatabus(addr++, rgb.r);
405         writeDatabus(addr++, rgb.g);
406         writeDatabus(addr++, rgb.b);
407     }
408 }
409 writeDatabus(RAM_INT_SEND, CMD_LEDS_SEND);
410 delay(1000);
411 }
412
413 void debugGameState()
414 {
415     printf(
416         "\nGame state: %d\nGame VS player: %d\nGame host: %s\n# colors:
417 %d\nCode: ",
418         game.state, game.vsPlayer, game.host->name, game.colors);
419     for (byte i = 0; i < COLORS; i++)
420         printf("%s ", ALL_COLOR_CLASSES[game.code[i]]);
421     printf("\n# of guesses: %d\nGuesses Table:\n", game.nrOfGuesses);
422     for (byte r = 0; r < ROWS; r++)
423     {
424         printf("Guess #%d: ", r);
425         for (byte c = 0; c < COLORS; c++)
426             printf("%10s ", ALL_COLOR_CLASSES[game.guesses[r][c]]);
427         printf(" Exact: %d Color: %d\n", game.guesses[r][COLORS],
428             game.guesses[r][COLORS + 1]);
429     }
430     printf("Code: ");
431     for (byte c = 0; c < COLORS; c++)
432         printf("%10s ", ALL_COLOR_CLASSES[game.code[c]]);
433     printf("\n\n");
434 }
435
436 void main()
437 {
438     // Get focus
439     BIOS_Set_Focus(FOCUS_APPLICATION);
440
441     /**
442     * INIT All of the things!
443     */
444     enableDatabus();
445
446     writeDatabus(RAM_INT_SEND, CMD_LCD_BL_ON);
447
448     writeDatabus(RAM_LEDS_DIM, 0xFF);
449     writeDatabus(RAM_LEDS_AMOUNT, ROWS * 2 * COLORS);
450
451     writeDatabus(RAM_LCD_CMD, 0x0C); // Blink off
452     writeDatabus(RAM_INT_SEND, CMD_LCD_CMD);
453
454     clearLCD();
455
456     printf("\n\nAVR firmware version id: %d 0x%02x\n\n",
457         readDatabus(RAM_VERSION_1), readDatabus(RAM_VERSION_2));
458     setLCDLine(0, "MMM by Dries007");
459     setLCDLine(1, "Bootng...");
460
461     // Ethernet connection check
462     if (BIOS_Ethernet_State(NULL, NULL))
463     {
464         setLCDLine(0, "ERROR");
465         setLCDLine(1, "NO ETHERNET!");
466
467         while (1)
468         {

```

```

469     test();
470 }
471
472     return;
473 }
474
475 /**
476  * RUN ALL TASKS
477  */
478 int result = RTX_Create_Task(&LCDupdateID, &LCDupdateTaskDefBlock);
479
480 if (result != 0)
481 {
482     printf("Creating/restart LCDupdate failed %d, exit program\n", result);
483     //delete task1
484     RTX_Delete_Task(LCDupdateID);
485     endProgram();
486 }
487
488 /**
489  * CGI methods
490  */
491
492 printf("Installing CGI methods\n");
493 installCGIMethods();
494
495 printf("Reset Game status\n");
496 resetGame();
497
498 /**
499  * MENU
500  */
501 byte key;
502 while (1)
503 {
504     printf("-~= Menu =~-~\n");
505     printf("-----\n");
506     printf("[X] End program\n");
507     printf("[R] Reboot\n");
508     printf("[D] Debug RAM Dump\n");
509     printf("[S] Set RAM manually\n");
510     printf("[G] Debug Game State\n");
511     printf("[U] Print all known users\n");
512     printf("[I] Interrupt test to AVR\n");
513
514     scanf("%c%c", &key);
515
516     switch (key & ~0x20)
517     {
518     case 'X':
519         endProgram();
520         break;
521     case 'R':
522         BIOS_Reboot();
523         break;
524     case 'D':
525         ramdump();
526         break;
527     case 'S':
528         manualram();
529         break;
530     case 'G':
531         debugGameState();
532         break;
533     case 'U':
534         printAllUsers();
535         break;

```

```

536     case 'I':
537         test();
538         break;
539     default:
540         printf("Char not in menu: %c\n", key);
541     }
542 }
543 }
544
545 /*****
546  *                               WEB
547  *****/
548
549 /*****
550  *                               TEMPLATE PARTS
551  *****/
552
553 char * pageHeader =
554     "<!doctype html><html><head><meta charset='us-ascii'><meta
555     name='viewport' content='width=400' /><title>Mastermind</title><link
556     href='http://fonts.googleapis.com/css?family=Open+Sans:600,400'
557     rel='stylesheet' type='text/css'><style type='text/css'>body,html{ font-
558     family:'Open Sans',sans-serif;margin:0 auto;padding:0;height:100%;min-
559     height:100%;position:relative;max-width:500px}#wrapper{padding:10px 10px
560     30px}.center{text-align:center}.black{background-
561     color:#000;color:#fff}.white{background-color:#fff}.orange{background-
562     color:orange}.purple{background-color:purple;color:#fff}.yellow{background-
563     color:#ff0}.aqua{background-color:#0ff}.pink{background-
564     color:#ff1493}.blue{background-color:#00f;color:#fff}.green{background-
565     color:green;color:#fff}.red{background-color:red}header h1{margin:1px}header
566     ul{margin:1px;padding:1px}header ul li{display:inline;padding:0
567     10px;margin:1px;border:1px solid #000;border-radius:5px;box-shadow:2px 2px
568     3px #888}header ul li a{tekst-
569     decoration:none;color:#000}.guesses{width:100%;text-align:center;;border-
570     collapse:collapse}.guesses .txt{padding:0 10px}.guesses * tr td{border:1px
571     solid #000}footer{position:absolute;bottom:0;height:30px;padding:0
572     10px}footer a{text-decoration:none;color:#d3d3d3;font-
573     size:smaller}input[type=submit], .btn{display:inline;padding:3px
574     10px;margin:1px;border:1px solid #000;border-radius:5px;box-shadow:2px 2px
575     3px #888;background:#fff;text-
576     decoration:none;color:#000}</style></head><body><div id='wrapper'>";
577
578 char * pageFooter =
579     "</div><footer><a class='center' href='http://www.dries007.net/'>&copy;
580     Dries007.net - 2015</a></footer></body></html>";
581
582 char * pickUsername =
583     "<h2>Welcome new player!</h2><p>Before you can play, you need to pick a
584     username:</p><form method='post' action='pickUsername'><input type='text'
585     name='username' maxlength='20'><input type='submit'
586     value='Check'></form>";
587
588 char * noGameYet = "<p>No game is going yet, but you can start one!</p>";
589
590 char * gameAvailable =
591     "<p>A game is currently being played. Go ahead an join!</p>";
592
593 /**
594  * Needs 2 extra strings per count. First string is URL, second is name.
595  */
596 void addMenuItems(char * buffer, byte count, ...)
597 {
598     strcat(buffer, "<header class='center'><h1>Mastermind</h1><ul>");
599
600     va_list ap;
601     va_start(ap, count);
602

```



```

603     for (byte i = 0; i < count; i++)
604     {
605         strcat(buffer, "<li><a href='");
606         strcat(buffer, va_arg(ap, char *));
607         strcat(buffer, "'>");
608         strcat(buffer, va_arg(ap, char *));
609         strcat(buffer, "</a></li>");
610     }
611
612     va_end(ap);
613     strcat(buffer, "</ul></header>");
614 }
615
616 /*****
617  *                               HOME (GET)
618  *****/
619
620 void huge _pascal _saveregs cgiHomeFunction(rpCgiPtr CgiRequest)
621 {
622     static char pageBuffer[2048]; // Buffer to contain web page
623     //char tmpBuffer[512]; // Buffer for string manipulation functions
624
625     sprintf(pageBuffer, pageHeader);
626
627     User * user = getUserByIP(CgiRequest->fRemoteIPPtr);
628     if (user == NULL)
629     {
630         addMenuItems(pageBuffer, 0);
631         strcat(pageBuffer, pickUsername);
632     }
633     else
634     {
635         if (getGame()->state == STATE_NO_GAME) addMenuItems(pageBuffer, 1,
636             "start", "Start a game");
637         else addMenuItems(pageBuffer, 1, "play", "Play");
638
639         strcat(pageBuffer, "<h2>Welcome ");
640         strcat(pageBuffer, user->name);
641         strcat(pageBuffer, "</h2>");
642
643         if (getGame()->state == STATE_NO_GAME) strcat(pageBuffer, noGameYet);
644         else strcat(pageBuffer, gameAvailable);
645     }
646
647     strcat(pageBuffer, pageFooter);
648
649     CgiRequest->fHttpResponse = CgiHttpOk;
650     CgiRequest->fDataType = CGIDataTypeHtml;
651     CgiRequest->fResponseBufferPtr = pageBuffer;
652     CgiRequest->fResponseBufferLength = strlen(pageBuffer);
653 }
654
655 /*****
656  *                               PICK USERNAME (POST)
657  *****/
658
659 void huge _pascal _saveregs cgiPickUsernameFunction(rpCgiPtr CgiRequest)
660 {
661     static char pageBuffer[2048]; // Buffer to contain web page
662     //char tmpBuffer[512]; // Buffer for string manipulation functions
663
664     sprintf(pageBuffer, pageHeader);
665
666     char * name;
667     char * value;
668     while (CGI_GetArgument(&name, &value, CgiRequest) == CGI_ARGUMENT_ERR_OK)
669     {

```

```

670     if (strcmp(name, "username") == 0)
671     {
672         if (getUserByName(value) != NULL)
673         {
674             addMenuItems(pageBuffer, 0);
675
676             strcat(pageBuffer, "<p>Sorry, ");
677             strcat(pageBuffer, value);
678             strcat(pageBuffer,
679                 " is already in use. Pick another name please:</p><form
680 method='post' action='pickUsername'><input type='text'
681 name='username'></input type='submit' value='Check'></form>");
682         }
683         else
684         {
685             addUser(*(long *) CgiRequest->fRemoteIPPtr, value);
686
687             if (getGame()->state == STATE_NO_GAME) addMenuItems(pageBuffer, 1,
688                 "start", "Start a game");
689             else addMenuItems(pageBuffer, 1, "play", "Play");
690
691             strcat(pageBuffer, "<p>You are now known as ");
692             strcat(pageBuffer, value);
693             strcat(pageBuffer, "!</p>");
694
695             if (getGame()->state == STATE_NO_GAME) strcat(pageBuffer,
696                 noGameYet);
697             else strcat(pageBuffer, gameAvailable);
698         }
699     }
700 }
701
702 strcat(pageBuffer, pageFooter);
703
704 CgiRequest->fHttpResponse = CgiHttpOk;
705 CgiRequest->fDataType = CGIDataTypeHtml;
706 CgiRequest->fResponseBufferPtr = pageBuffer;
707 CgiRequest->fResponseBufferLength = strlen(pageBuffer);
708 }
709
710 /*****
711  *          START (BOTH)
712  *****/
713
714 void huge _pascal _saveregs cgiStartFunction(rpCgiPtr CgiRequest)
715 {
716     static char pageBuffer[2048]; // Buffer to contain web page
717     char tmpBuffer[512]; // Buffer for string manipulation functions
718
719     sprintf(pageBuffer, pageHeader);
720
721     User * user = getUserByIP(CgiRequest->fRemoteIPPtr);
722     Game * game = getGame();
723     if (user == NULL)
724     {
725         addMenuItems(pageBuffer, 0);
726         strcat(pageBuffer, pickUsername);
727     }
728     else if (game->state == STATE_NO_GAME
729         || game->state == STATE_GAME_CONFIGURED) // If game is not (fully)
730         configured yet
731     {
732         char * name;
733         char * value;
734         while (CGI_GetArgument(&name, &value, CgiRequest)
735             == CGI_ARGUMENT_ERR_OK) // Argument parse loop
736         {

```

```

737     if (strcmp(name, "mode") == 0) // Gamemode
738     {
739         game->vsPlayer = strcmp(value, "Player") == 0;
740         game->state = STATE_GAME_CONFIGURED;
741     }
742     else if (strcmp(name, "colors") == 0) // # of colors
743     {
744         game->colors = atoi(value);
745         game->state = STATE_GAME_CONFIGURED;
746     }
747     else // Colors of code
748     {
749         int i;
750         sscanf(name, "c%d", &i);
751         game->code[i] = atoi(value);
752         game->state = STATE_GAME_STARTED;
753     }
754 }
755
756 if (game->state == STATE_GAME_CONFIGURED) // If game is partially
757 configured (argument parser above)
758 {
759     if (game->vsPlayer) // Print color picker code
760     {
761         game->host = user;
762         addMenuItems(pageBuffer, 0);
763         strcat(pageBuffer,
764             "<p>Pick your code:</p><form method='get' action='start'>");
765         for (byte i = 0; i < 4; i++) // Color picker 1 -> 4
766         {
767             sprintf(tmpBuffer, "<select name='c%d'", i);
768             strcat(pageBuffer, tmpBuffer);
769
770             for (byte c = 0; c < game->colors; c++)
771             {
772                 sprintf(tmpBuffer, "<option value='%d' class='%s'>%s</option>",
773                     c, ALL_COLOR_CLASSES[c], ALL_COLOR_CLASSES[c]);
774                 strcat(pageBuffer, tmpBuffer);
775             }
776
777             strcat(pageBuffer, "</select>");
778         }
779         strcat(pageBuffer,
780             "<input type='submit' value='Choose!'></form>");
781     }
782     else // VS computer
783     {
784         addMenuItems(pageBuffer, 1, "play", "Play");
785         strcat(pageBuffer, "<h2>Game started!</h2>");
786         setRndCode(game->colors);
787         game->state = STATE_GAME_STARTED;
788     }
789 }
790 else if (game->state == STATE_GAME_STARTED) // Game started
791 {
792     addMenuItems(pageBuffer, 1, "play", "Play");
793     strcat(pageBuffer, "<h2>Game started!</h2>");
794 }
795 else // New game form
796 {
797     addMenuItems(pageBuffer, 0);
798     strcat(pageBuffer,
799         "<h2>Start a new game</h2><form method='get' action='start'><p>
800 Player(s) VS <label><input type='radio' name='mode' value='Player' checked/>
801 Host</label><label><input type='radio' name='mode' value='Computer'>
802 Computer</label></p><p><label for='colors'># of colors: </label><select
803 id='colors'

```

```

804     name='colors'><option>4</option><option>6</option><option>8</option></select>
805     </p><input type='submit' value='Go!' /></form>");
806     }
807 }
808 else // Already going
809 {
810     addMenuItems(pageBuffer, 1, "play", "Play");
811     strcat(pageBuffer,
812         "<h2>Start a new game</h2><p>A game has already been started.</p>");
813 }
814
815 strcat(pageBuffer, pageFooter);
816
817 CgiRequest->fHttpResponse = CgiHttpOk;
818 CgiRequest->fDataType = CGIDataTypeHtml;
819 CgiRequest->fResponseBufferPtr = pageBuffer;
820 CgiRequest->fResponseBufferLength = strlen(pageBuffer);
821 }
822
823 /*****
824  *                                PLAY (GET)
825  *****/
826
827 void huge _pascal _saveregs cgiPlayFunction(rpCgiPtr CgiRequest)
828 {
829     static char pageBuffer[2048]; // Buffer to contain web page
830     char tmpBuffer[512]; // Buffer for string manipulation functions
831
832     sprintf(pageBuffer, pageHeader);
833
834     User * user = getUserByIP(CgiRequest->fRemoteIPPtr);
835     Game * game = getGame();
836
837     if (user == NULL)
838     {
839         addMenuItems(pageBuffer, 0);
840         strcat(pageBuffer, pickUsername);
841     }
842     else if (game->state == STATE_NO_GAME) // If game is not configured yet
843     {
844         addMenuItems(pageBuffer, 1, "start", "Start a game");
845         strcat(pageBuffer, noGameYet);
846     }
847     else
848     {
849         addMenuItems(pageBuffer, 0);
850
851         if (game->state == STATE_GAME_STARTED)
852         {
853             // Process guess, if any
854             char * name;
855             char * value;
856             byte i = 0xFF;
857             while (CGI_GetArgument(&name, &value, CgiRequest)
858                 == CGI_ARGUMENT_ERR_OK)
859             {
860                 int p, c; //p = position, c = color id
861                 sscanf(name, "c%d", &p);
862                 c = atoi(value);
863                 if (i == 0xFF) i = game->nrOfGuesses++;
864                 game->guesses[i][p] = c;
865             }
866             if (i != 0xFF) guessRow(i);
867         }
868
869         // Display guess table
870

```

```

871     strcat(pageBuffer,
872         "<p>Guesses:</p><table class='guesses' border> <tr> <th
873 class='txt'>#</th>");
874     for (byte c = 0; c < COLORS; c++)
875         strcat(pageBuffer, "<th style='min-width: 50px;'></th>");
876     strcat(pageBuffer,
877         "<th class='txt'>Exact</th> <th class='txt'>Color</th> </tr>");
878
879     for (byte i = 0; i < game->nrOfGuesses; i++)
880     {
881         sprintf(tmpBuffer, "<tr><td style='padding: 0 10px'>%d</td>", i);
882         strcat(pageBuffer, tmpBuffer);
883
884         for (byte c = 0; c < COLORS; c++)
885         {
886             char * color = ALL_COLOR_CLASSES[game->guesses[i][c]];
887             sprintf(tmpBuffer, "<td class='%s'>%s</td>", color, color);
888             strcat(pageBuffer, tmpBuffer);
889         }
890         sprintf(tmpBuffer, "<td>%d</td><td>%d</td></tr>",
891             game->guesses[i][COLORS], game->guesses[i][COLORS + 1]);
892         strcat(pageBuffer, tmpBuffer);
893     }
894
895     strcat(pageBuffer, "</table>");
896
897     if (game->state == STATE_GAME_OVER)
898     {
899         strcat(pageBuffer,
900             "<p>Game over! The host/computer won!</p><a href='reset'
901 class='btn'>Reset</a>");
902     }
903     else if (game->state == STATE_GAME_WON)
904     {
905         strcat(pageBuffer,
906             "<p>Game over! The codebreaker(s) won!</p><a href='reset'
907 class='btn'>Reset</a>");
908     }
909     else if (game->vsPlayer && game->host == user)
910     {
911         strcat(pageBuffer, "<p>You picked the code, you can't guess.</p>");
912     }
913     else if (game->state == STATE_GAME_STARTED) // Let user make guess
914     {
915         strcat(pageBuffer,
916             "<p>Make a guess:</p><form method='get' action='play'>");
917         for (byte i = 0; i < COLORS; i++) // Color picker 1 -> 4
918         {
919             sprintf(tmpBuffer, "<select name='c%d'>", i);
920             strcat(pageBuffer, tmpBuffer);
921
922             for (byte c = 0; c < game->colors; c++)
923             {
924                 byte selected = game->nrOfGuesses != 0
925                     && game->guesses[game->nrOfGuesses - 1][i] == c;
926                 sprintf(tmpBuffer,
927                     "<option value='%d' class='%s' %s %s></option>", c,
928                     ALL_COLOR_CLASSES[c], selected ? "selected" : "",
929                     ALL_COLOR_CLASSES[c]);
930                 strcat(pageBuffer, tmpBuffer);
931             }
932
933             strcat(pageBuffer, "</select>");
934         }
935         strcat(pageBuffer, "<input type='submit' value='Choose!'></form>");
936     }
937     else

```

```

938     {
939         strcat(pageBuffer,
940             "<p>Code is not yet picked. Refresh the page to get a status
941 update.</p>");
942     }
943 }
944 strcat(pageBuffer, pageFooter);
945
946 CgiRequest->fHttpResponse = CgiHttpOk;
947 CgiRequest->fDataType = CGIDataTypeHtml;
948 CgiRequest->fResponseBufferPtr = pageBuffer;
949 CgiRequest->fResponseBufferLength = strlen(pageBuffer);
950 }
951
952 /*****
953  *          RESET (GET)
954  *****/
955
956 void huge _pascal _saveregs cgiResetFunction(rpCgiPtr CgiRequest)
957 {
958     static char pageBuffer[2048]; // Buffer to contain web page
959     char tmpBuffer[512]; // Buffer for string manipulation functions
960
961     sprintf(pageBuffer, pageHeader);
962
963     User * user = getUserByIP(CgiRequest->fRemoteIPPtr);
964     Game * game = getGame();
965
966     if (user == NULL)
967     {
968         addMenuItems(pageBuffer, 0);
969         strcat(pageBuffer, pickUsername);
970     }
971     else if (game->state == STATE_GAME_OVER || game->state == STATE_GAME_WON)
972     // If game done
973     {
974         resetGame();
975
976         addMenuItems(pageBuffer, 1, "start", "Start a game");
977         strcat(pageBuffer, "<p>The game has been reset.</p>");
978     }
979     else
980     {
981         strcat(pageBuffer, "<p>Incorrect game state.</p>");
982     }
983
984     strcat(pageBuffer, pageFooter);
985
986     CgiRequest->fHttpResponse = CgiHttpOk;
987     CgiRequest->fDataType = CGIDataTypeHtml;
988     CgiRequest->fResponseBufferPtr = pageBuffer;
989     CgiRequest->fResponseBufferLength = strlen(pageBuffer);
990 }
991
992 /*****
993  *          ALL INSTALL / REMOVE LOGIC
994  *****/
995
996 typedef void huge _pascal _saveregs (*CGIfn)(rpCgiPtr); // Because function
997 pointer syntax in unreadable
998
999 char *cgiNames[] = { "home", "pickUsername", "start", "play", "reset" };
1000 int cgiMethods[] = { CgiHttpGet, CgiHttpPost, CgiHttpGet, CgiHttpGet,
1001     CgiHttpGet };
1002 CGIfn cgiFunctions[] = { cgiHomeFunction, cgiPickUsernameFunction,
1003     cgiStartFunction, cgiPlayFunction, cgiResetFunction };
1004

```

```

1005 void installCGIMethods()
1006 {
1007     CGI_Entry cgiEntry;
1008
1009     for (byte i = 0; i < 5; i++)
1010     {
1011         cgiEntry.PathPtr = cgiNames[i];
1012         cgiEntry.CgiFuncPtr = cgiFunctions[i];
1013         cgiEntry.method = cgiMethods[i];
1014
1015         if (CGI_Install(&cgiEntry) != 0)
1016         {
1017             printf("Installing CGI function %s failed\n", cgiEntry.PathPtr);
1018             endProgram();
1019         }
1020     }
1021 }
1022
1023 void removeCGIMethods()
1024 {
1025     byte n = sizeof(cgiMethods) / sizeof(int);
1026     for (byte i = 0; i < n; i++)
1027     {
1028         if (CGI_Delete(cgiNames[i]))
1029         {
1030             printf("Removing %s failed\n", cgiNames[i]);
1031         }
1032     }
1033 }

```

### 6.2.3 HTML template

Dit is de niet gecondenseerde versie van de HTML code gebruikt in *mastermind.c* regel 554 t.e.m. 576.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset='us-ascii'>
5          <meta content='width=400' name='viewport'>
6          <title>
7              Mastermind
8          </title>
9          <link
10 href='http://fonts.googleapis.com/css?family=Open+Sans:600,400'
11 rel='stylesheet' type='text/css'>
12          <style type='text/css'>
13              body,html {
14                  font-family:'Open Sans',sans-serif;
15                  margin:0 auto;
16                  padding:0;
17                  height:100%;
18                  min-height:100%;
19                  position:relative;
20                  max-width:500px
21              }
22              header ul li {
23                  display:inline;
24                  padding:0 10px;
25                  margin:1px;
26                  border:1px solid #000;
27                  border-radius:5px;
28                  box-shadow:2px 2px 3px #888
29              }
30              guesses {
31                  width:100%;
32                  text-align:center;
33                  border-collapse:collapse
34              }
35              footer {
36                  position:absolute;
37                  bottom:0;
38                  height:30px;
39                  padding:0 10px
40              }
41              footer a {
42                  text-decoration:none;
43                  color:#d3d3d3;
44                  font-size:smaller
45              }
46              input[type=submit],.btn {
47                  display:inline;
48                  padding:3px 10px;
49                  margin:1px;
50                  border:1px solid #000;
51                  border-radius:5px;
52                  box-shadow:2px 2px 3px #888;
53                  background:#fff;
54                  text-decoration:none;
55                  color:#000
56              }
57              #wrapper { padding:10px 10px 30px }
58              .center { text-align:center }
59              .black { background-color:#000; color:#fff }
60              .white { background-color:#fff }
61              .orange { background-color:orange }
```



```

62         .purple { background-color:purple; color:#fff }
63         .yellow { background-color:#ff0 }
64         .aqua { background-color:#0ff }
65         .pink { background-color:#ff1493 }
66         .blue { background-color:#00f; color:#fff }
67         .green { background-color:green; color:#fff }
68         .red { background-color:red }
69         header h1 { margin:1px }
70         header ul { margin:1px; padding:1px }
71         header ul li a { tekst-decoration:none; color:#000 }
72         guesses .txt { padding:0 10px }
73         guesses * tr td { border:1px solid #000 }
74     </style>
75 </head>
76 <body>
77     <div id='wrapper'></div>
78     <footer>
79         <a class='center' href='http://www.dries007.net/'>&copy;
80 Dries007.net - 2015</a>
81     </footer>
82 </body>
83 </html>

```

## 6.2.4 AVR.h

```
1  #ifndef AVR_H_
2  #define AVR_H_
3
4  #define DEBUG 0
5
6  /* ===== PORT CONFIG ===== */
7  // Mask to eliminate INT4
8  #define LCD_PORT PORTB
9  #define LCD_DDR DDRB
10 // KP = Keypad
11 #define KP_PORT PORTD
12 #define KP_DDR DDRD
13 #define KP_PIN PIND
14 // WS2812
15 #define LEDS_PORT PORTE
16 #define LEDS_DDR DDRE
17 #define LEDS_PIN 7
18
19 /* ===== RAM ADDRESSES ===== */
20 // Offset for External RAM
21 #define RAM_OFFSET 0x8000
22 // 170 Leds max (x 3 bytes = 0x1FE)
23 #define RAM_LEDS_START (RAM_OFFSET + 0x000)
24 #define RAM_LEDS_END (RAM_OFFSET + 0x1FE)
25 #define RAM_LEDS_AMOUNT (RAM_OFFSET + 0x1FF)
26 // 80 bytes of char buffer for LCD
27 #define RAM_LCD_START (RAM_OFFSET + 0x200)
28 #define RAM_LCD_END (RAM_OFFSET + 0x250)
29 #define RAM_LCD_CMD (RAM_OFFSET + 0x251)
30 // Mask for global LED dimming
31 #define RAM_LEDS_DIM (RAM_OFFSET + 0x252)
32 #define RAM_KP_LASTKEY (RAM_OFFSET + 0x253)
33
34 #define RAM_VERSION_1 (RAM_OFFSET + 0x300)
35 #define RAM_VERSION_2 (RAM_OFFSET + 0x301)
36
37 // Interrupt registers
38 #define RAM_INT_SEND (RAM_OFFSET + 0x3FE)
39 #define RAM_INT_GET (RAM_OFFSET + 0x3FF)
40
41 #define MAX_LEDS ((uint8_t)(RAM_LEDS_END - RAM_LEDS_START))
42 #define MAX_LCD_CHARS ((uint8_t)(RAM_LCD_END - RAM_LCD_START))
43
44 /* ===== CMD CODES ===== */
45 #define CMD_LEDS_SEND 0x01 // Clock out LEDS
46 #define CMD_LCD_CHAR 0x02 // Print char buffer to LCD
47 #define CMD_LCD_CMD 0x03 // Send instruction byte to LCD
48 #define CMD_LCD_CL_PR 0x04 // Send clear + print out char buffer to LCD
49 #define CMD_LCD_POS 0x05 // Set LCD cursor to position
50 #define CMD_LCD_BL_ON 0x06 // Set LCD_BACKLIGHT = 1
51 #define CMD_LCD_BL_OFF 0x07 // Set LCD_BACKLIGHT = 0
52
53 #define STATUS_KP_PRESS 0x01 // Key was pressed
54
55 /* ===== MACROS ===== */
56 /*
57  !WARNING! The data-lines on my custom PCB are flipped on the AVR side.
58  This means that these macros use the '__builtin_avr_insert_bits' macro to
59  flip the data read/written from/to the DP-RAM.
60  If you don't need this, replace '__builtin_avr_insert_bits (0x01234567,
61  {value}, 0)' with '{value}'.
62  */
63 #define PONTER_RAM(addr) ( ((volatile uint8_t *) addr) )
64 #define READ_RAM(addr) ( __builtin_avr_insert_bits (0x01234567,
65 *PONTER_RAM(addr), 0) )
66 #define WRITE_RAM(addr, val) { *PONTER_RAM(addr) =
```

```

67 __builtin_avr_insert_bits (0x01234567, val, 0); }
68
69 /* ===== STRUCTS ===== */
70 // Correct byte order!
71 struct cRGB { uint8_t g; uint8_t r; uint8_t b; };
72
73 /* ===== GLOBALS ===== */
74 struct cRGB LEDS[MAX_LEDS]; // LED data, in correct byte order
75 uint8_t LCD_BACKLIGHT = 0; // LCD back light pin status (pin 7)
76
77 /* ===== FUNCTIONS ===== */
78
79 /* ----- inits -----
80 * Parameters: N/a
81 * Actions:
82 * - Enable External Memory Interface
83 * - Set `LCD_PORT` as output (with `LCD_MASK`)
84 * - Set `KP_PORT`.[0->3] as output
85 * - Set `KP_PORT`.[4->8] as input
86 * - Set `LEDS_PORT`.\`LEDS_PIN` as output
87 * - Init & Clear LCD
88 * - Backlight LCD ON
89 * - LCD Cursor OFF
90 */
91 void inline inits();
92
93 /* ----- readMatrix -----
94 * Parameters:
95 * - matrix          Pin readout of the matrix port (1 = active, invert
96 when using pull-ups)
97 * Actions:
98 * - Convert matrix code to ACSII
99 * Matrix layout:
100 *   Bit | 5 | 6 | 7 | 8
101 *   ----+---+---+---+---
102 *   1 | 1 | 2 | 3 | A
103 *   2 | 4 | 5 | 6 | B
104 *   3 | 7 | 8 | 9 | C
105 *   4 | * | 0 | # | D
106 * Pressing 2 buttons at once results in 0x00, same as no button pressed
107 */
108 uint8_t inline readMatrix(uint8_t matrix);
109
110 /* ----- sendLEDS -----
111 * Parameters:
112 * - amountOfLeds    The amount of led information to send
113 * Actions:
114 * - Send LED data in `LEDS` to `LEDS_PORT`.\`LEDS_PIN`
115 * - Delay 50µs for reset pulse
116 * Prerequisites:
117 * - `LEDS_PORT`.\`LEDS_PIN` has been set as output
118 */
119 void inline sendLEDS(uint16_t amountOfLeds);
120
121 /* ----- sendLCDNibble -----
122 * Parameters:
123 * - data            Populate 4 lower bits
124 * - rs              1 for character, 0 for instruction
125 * Actions:
126 * - Mask data with 0x0F
127 * - Set data bit 7 (Backlight) to `LCD_BACKLIGHT`
128 * - Set data bit 6 (RegisterSelect) to `rs`
129 * - Set data bit 5 (Enable) to 1
130 * - Set data bit 4 (INT4) to 1 (for Pull-up)
131 * - Write data to `LCD_PORT`
132 * - delay 1 µs
133 * - Toggle bit 5 (Enable)

```

```

134     * - delay 2 µs
135 */
136 void sendLCDNibble(uint8_t data, uint8_t rs);
137
138 /* ----- sendLCDInstructionByte -----
139 * Parameters:
140 * - data          Data to send to the instruction register
141 * Actions:
142 * - Call sendLCDNibble(data >> 4, 0)
143 * - Call sendLCDNibble(data, 0)
144 * - delay 50µs
145 */
146 void sendLCDInstructionByte(uint8_t data);
147
148 /* ----- sendLCDCharacterByte -----
149 * Parameters:
150 * - data          Data to send to the character register
151 * Actions:
152 * - Call sendLCDNibble(data >> 4, 1)
153 * - Call sendLCDNibble(data, 1)
154 * - delay 50µs
155 */
156 void sendLCDCharacterByte(char data);
157
158 /* ----- sendLCDBuffer -----
159 * Parameters:
160 * - *buffer       Pointer to character buffer
161 * Actions:
162 * - For every character in the buffer or until `MAX_LCD_CHARS` is reached:
163 *   - Call sendLCDCharacterByte(data)
164 */
165 void sendLCDBuffer(char * buffer);
166
167 /* ***** START SECTION WS2812 DRIVER *****
168 * Original Source:  https://github.com/cpldcpu/light\_ws2812/
169 * Original Author:  Tim (cpldcpu@gmail.com)
170 * Original License: GNU GPL V2
171 (https://github.com/cpldcpu/light\_ws2812/blob/master/License.txt)
172 * This license still applies to everything between the "WS2812 DRIVER"
173 section lines.
174 *
175 * Modifications by Dries007:
176 * - Changed configuration
177 * - Merged library into single set of source and header files
178 */
179 // Timing in ns
180 #define w_zeropulse 350
181 #define w_onepulse 900
182 #define w_totalperiod 1250
183
184 // Fixed cycles used by the inner loop
185 #define w_fixedlow 2
186 #define w_fixedhigh 4
187 #define w_fixedtotal 8
188
189 // Insert NOPs to match the timing, if possible
190 #define w_zerocycles (((F_CPU/1000)*w_zeropulse)/1000000)
191 #define w_onecycles (((F_CPU/1000)*w_onepulse +500000)/1000000)
192 #define w_totalcycles (((F_CPU/1000)*w_totalperiod +500000)/1000000)
193
194 // w1 - nops between rising edge and falling edge - low
195 #define w1 (w_zerocycles-w_fixedlow)
196 // w2 nops between fe low and fe high
197 #define w2 (w_onecycles-w_fixedhigh-w1)
198 // w3 nops to complete loop
199 #define w3 (w_totalcycles-w_fixedtotal-w1-w2)
200

```

```

201  #if w1>0
202      #define w1_nops w1
203  #else
204      #define w1_nops 0
205  #endif
206
207  // The only critical timing parameter is the minimum pulse length of the "0"
208  // Warn or throw error if this timing can not be met with current F_CPU
209  settings.
210  #define w_lowtime ((w1_nops+w_fixedlow)*1000000)/(F_CPU/1000)
211  #if w_lowtime>550
212      #error "WS2812 DRIVER: Sorry, the clock speed is too low. Did you set
213  F_CPU correctly?"
214  #elif w_lowtime>450
215      #warning "WS2812 DRIVER: The timing is critical and may only work on
216  WS2812B, not on WS2812(S).\"
217      #warning "Please consider a higher clockspeed, if possible"
218  #endif
219
220  #if w2>0
221      #define w2_nops w2
222  #else
223      #define w2_nops 0
224  #endif
225
226  #if w3>0
227      #define w3_nops w3
228  #else
229      #define w3_nops 0
230  #endif
231
232  #define w_nop1  "nop      \n\t"
233  #define w_nop2  "rjmp .+0 \n\t"
234  #define w_nop4  w_nop2 w_nop2
235  #define w_nop8  w_nop4 w_nop4
236  #define w_nop16 w_nop8 w_nop8
237
238  /* ***** END SECTION WS2812 DRIVER ***** */
239
240  #endif /* AVR_H_ */

```

### 6.2.5 AVR.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <avr/interrupt.h>
4  #include <avr/io.h>
5  #include <avr/pgmspace.h>
6  #include <util/delay.h>
7
8  #include "AVR.h"
9
10 /* ===== INTERRUPT SERVICE ROUTINES ===== */
11 // Interrupt Service Routine for INT4
12 ISR(INT4_vect)
13 {
14     // read interrupt address, also clears interrupt signal
15     volatile uint8_t cmd = READ_RAM(RAM_INT_GET);
16
17     switch (cmd)
18     {
19         // Copy LED data from DPRAM into RAM (sets correct byte order) and
20         // clock out the data
21         case CMD_LEDS_SEND:
22         {
23             // Amount of LEDS connected
24             uint8_t n = READ_RAM(RAM_LEDS_AMOUNT);
25             // Global LED dimmer settings
26             uint8_t dim = READ_RAM(RAM_LEDS_DIM);
27             // Make sure that n <= MAX_LEDS to prevent data corruption
28             if (n > MAX_LEDS) n = MAX_LEDS;
29             // used for DPRAM address offset from RAM_LEDS_START
30             uint16_t offset = 0;
31             for (uint16_t i = 0; i < n; i++)
32             {
33                 LEDS[i].r = READ_RAM(RAM_LEDS_START + (offset ++)) & dim;
34                 LEDS[i].g = READ_RAM(RAM_LEDS_START + (offset ++)) & dim;
35                 LEDS[i].b = READ_RAM(RAM_LEDS_START + (offset ++)) & dim;
36             }
37             // Clock out data
38             sendLEDS(n);
39         }
40         break;
41         // Shortcut command to set LCD cursor position
42         case CMD_LCD_POS:
43         {
44             sendLCDInstructionByte(READ_RAM(RAM_LCD_CMD) | 0b10000000);
45         }
46         break;
47         // Send LCD an insrtuction byte
48         case CMD_LCD_CMD:
49         {
50             sendLCDInstructionByte(READ_RAM(RAM_LCD_CMD));
51         }
52         break;
53         // Shortcut for clear & print
54         case CMD_LCD_CL_PR:
55         {
56             sendLCDInstructionByte(0x01);
57             _delay_ms(10);
58         }
59         // no break!
60         // Print char buffer (until max chars or 0x00)
61         case CMD_LCD_CHAR:
62         {
63             for (uint8_t i = 0; i < MAX_LCD_CHARS; i++)
64             {
65                 uint8_t c = READ_RAM(RAM_LCD_START + i);
66                 if (c == 0x00) break;
```

```

67         sendLCDCharacterByte(c);
68     }
69 }
70 break;
71 case CMD_LCD_BL_ON: LCD_BACKLIGHT = 1; break;
72 case CMD_LCD_BL_OFF: LCD_BACKLIGHT = 0; break;
73 }
74 /* Interrupt detection debug code */
75 #if DEBUG
76     char buff[10];
77     sprintf(buff, "I:0x%02X", cmd);
78     sendLCDBuffer(buff);
79 #endif
80 sei();
81 }
82
83 /* ===== DEBUG STUFF ===== */
84 /* HANDLE DEBUG KEYPRESS HERE */
85 void debugKeypress(uint8_t key)
86 {
87     // buffer index pointer
88     static uint8_t b = 0;
89     // buffer (20 chars = 1 line)
90     static char buffer[20];
91
92     switch (key)
93     {
94         // SEND
95         case '*':
96         {
97             sendLEDS(30);
98
99             b = 0;
100             buffer[b] = 0;
101             sendLCDInstructionByte(0x01);
102             _delay_ms(2);
103             sprintf(buffer, "      0x%02X 0x%02X 0x%02X", LEDS[0].r,
104 LEDS[0].g, LEDS[0].b);
105             sendLCDBuffer(buffer);
106             // 1e pos on lcd
107             sendLCDInstructionByte(0x80);
108             break;
109         }
110         // BACKSPACE
111         case '#':
112         {
113             if (b != 0) b--;
114             buffer[b] = 0;
115
116             sendLCDInstructionByte(0x01);
117             _delay_ms(2);
118
119             sendLCDBuffer(buffer);
120             break;
121         }
122         // SET RED
123         case 'A':
124         {
125             uint8_t nr = atoi(buffer);
126             for (uint8_t i = 0; i < 30; i++)
127             {
128                 LEDS[i].r = nr;
129             }
130             b = 0;
131             buffer[b] = 0;
132             sendLCDInstructionByte(0x01);
133             _delay_ms(2);

```

```

134         break;
135     }
136     // SET GREEN
137     case 'B':
138     {
139         uint8_t nr = atoi(buffer);
140         for (uint8_t i = 0; i < 30; i++)
141         {
142             LEDS[i].g = nr;
143         }
144         b = 0;
145         buffer[b] = 0;
146         sendLCDInstructionByte(0x01);
147         _delay_ms(2);
148         break;
149     }
150     // SET BLUE
151     case 'C':
152     {
153         uint8_t nr = atoi(buffer);
154         for (uint8_t i = 0; i < 30; i++)
155         {
156             LEDS[i].b = nr;
157         }
158         b = 0;
159         buffer[b] = 0;
160         sendLCDInstructionByte(0x01);
161         _delay_ms(2);
162         break;
163     }
164     case 'D':
165     {
166         sendLCDInstructionByte(0x01);
167         for (uint8_t i = 0; i < MAX_LCD_CHARS; i++)
168         {
169             uint8_t c = READ_RAM(RAM_LCD_START + i);
170             if (c == 0x00) break;
171             sendLCDCharacterByte(c);
172         }
173     }
174     break;
175     // NUMBER
176     default:
177     {
178         buffer[b++] = key;
179         buffer[b] = 0;
180         sendLCDCharacterByte(key);
181         break;
182     }
183 }
184 }
185
186 /* ===== MAIN ===== */
187 int main()
188 {
189     // set ports & interrupt registers
190     inits();
191
192     #if DEBUG
193     // debounce variables
194     uint8_t prevKey = 0x00;
195     uint16_t downTime = 0;
196     uint16_t upTime = 0;
197
198     // debug program loop (aka keypad scanner)
199     while (1)
200     {

```



```

201     // ROW loop
202     for (uint8_t r = 0; r < 4; r++)
203     {
204         // all pins HIGH, the row we want to read LOW; bit 0-4 always high
205         because they are inputs (pull-up).
206         KP_PORT = 0x0F | (0b11101111 << r);
207         // Convert read port byte (inverted because pull-ups)
208
209         uint8_t key = readMatrix(~KP_PIN);
210         // no key pressed
211         if (key == 0x00)
212         {
213             // if no key was pressed for 100+ ms, reset debounce.
214             if (upTime++ > 100)
215             {
216                 // Makes sure the next keypress will register instantly
217                 prevKey = 0x00;
218
219                 downTime = 0;
220                 upTime = 0;
221             }
222         }
223         else // A key was pressed
224         {
225             // if the pressed key is different from the last one OR its
226             been pressed for 500+ ms, acknowledge as a legitimate press
227             if (prevKey != key || downTime++ > 500)
228             {
229                 // Store key in DPRAM for SC12
230                 WRITE_RAM(RAM_KP_LASTKEY, key);
231                 // Send interrupt to SC12
232                 WRITE_RAM(RAM_INT_SEND, SATUS_KP_PRESS);
233
234                 debugKeypress(key);
235
236                 // Store current key for debounce
237                 prevKey = key;
238
239                 downTime = 0;
240                 upTime = 0;
241             }
242         }
243
244         _delay_ms(1);
245     }
246 }
247 #else
248 while (1)
249 {
250
251 }
252 #endif
253 }
254
255 /* ===== FUNCTIONS ===== */
256 void inline inits()
257 {
258     // Write SRE to 1 enables the External Memory Interface
259     MCUCR = 0x80;
260
261     // Magic numbers
262     WRITE_RAM(RAM_VERSION_1, 42);
263     WRITE_RAM(RAM_VERSION_2, 0x42);
264
265     // LCD Port Setup
266     LCD_DDR = 0xFF;
267

```

```

268 // Keypad Port Setup (bit 0-3 = in; bit 4-7 = out)
269 KP_DDR = 0xF0;
270
271 // LED port all output
272 LEDS_DDR = 0xEF;
273 LEDS_PORT = (uint8_t)~0xEF;
274
275 // Enable falling edge interrupt INT4
276 EICRB = 0x02;
277 EIMSK = 0x10;
278
279 // LCD init
280 _delay_ms(100);
281
282 // Set 4 bit mode
283 sendLCDNible(0x02, 0);
284 // 2-line mode, display on
285 sendLCDInstructionByte(0x0C);
286 _delay_ms(100);
287
288 // Display ON/OFF Control
289 sendLCDInstructionByte(0x0F);
290 // Clear & home
291 sendLCDInstructionByte(0x01);
292 _delay_ms(20);
293 // Entry mode Increment & Entire shift off
294 sendLCDInstructionByte(0x06);
295
296 // Clear any open interrupts.
297 volatile uint8_t i = READ_RAM(RAM_INT_GET);
298 // Global interrupts ON
299 sei();
300 }
301
302 uint8_t inline readMatrix(uint8_t matrix)
303 {
304     switch (matrix)
305     {
306         default: return 0x00;
307
308         case 0b00010001: return '1'; // 0x11
309         case 0b00010010: return '4'; // 0x12
310         case 0b00010100: return '7'; // 0x13
311         case 0b00011000: return '*'; // 0x18
312
313         case 0b00100001: return '2'; // 0x21
314         case 0b00100010: return '5'; // 0x22
315         case 0b00100100: return '8'; // 0x24
316         case 0b00101000: return '0'; // 0x28
317
318         case 0b01000001: return '3'; // 0x41
319         case 0b01000010: return '6'; // 0x42
320         case 0b01000100: return '9'; // 0x44
321         case 0b01001000: return '#'; // 0x48
322
323         case 0b10000001: return 'A'; // 0x81
324         case 0b10000010: return 'B'; // 0x82
325         case 0b10000100: return 'C'; // 0x84
326         case 0b10001000: return 'D'; // 0x88
327     }
328 }
329
330 /* ***** START SECTION WS2812 DRIVER *****
331 * Original Source:   https://github.com/cpldcpu/light_ws2812/
332 * Original Author:   Tim (cpldcpu@gmail.com)
333 * Original License:  GNU GPL V2
334 (https://github.com/cpldcpu/light_ws2812/blob/master/License.txt)

```

```

335  *      This license still applies to everything between the "WS2812 DRIVER"
336  section lines.
337  *
338  * Modifications by Dries007:
339  * - Changed configuration
340  * - Merged library into single set of source and header files
341  */
342 void inline sendLEDS(uint16_t leds)
343 {
344     // 3 colors!
345     uint16_t datlen = leds + leds + leds;
346     // Type cast
347     uint8_t * data = (uint8_t *) LEDS;
348
349     // Save interrupt status
350     uint8_t sreg_prev = SREG;
351     // We can't be interrupted!
352     cli();
353
354     uint8_t maskhi = _BV(LED_PIN);
355     // Low mask
356     uint8_t masklo = ~maskhi & LED_PORT;
357     // High mask
358     maskhi |= LED_PORT;
359
360     // used in ASM
361     uint8_t curbyte, ctr;
362     while (datlen--)
363     {
364         curbyte = *data ++; // Grab byte
365
366         asm volatile(
367             "        ldi    %0,8  \n\t" // Write 8 (00001000) to Loop counter (%0)
368             "loop%=:      \n\t" // Loop entry point (%= is a unique number
on each asm statement)
369             "        out    %2,%3 \n\t" // Write High mask (%3) to LED_PORT (%2)
370             #if (w1_nops&1) // w1 nops for timing
371             w_nop1
372             #endif
373             #if (w1_nops&2)
374             w_nop2
375             #endif
376             #if (w1_nops&4)
377             w_nop4
378             #endif
379             #if (w1_nops&8)
380             w_nop8
381             #endif
382             #if (w1_nops&16)
383             w_nop16
384             #endif
385             "        sbrs   %1,7  \n\t" // Skip next instruction if bit 7 of Data
386             "(%1) is set
387             "        out    %2,%4 \n\t" // Write Low mask (%4) to LED_PORT (%2)
388             "        lsl    %1    \n\t" // Shift Data (%1) left
389             #if (w2_nops&1) // w2 nops for timing
390             w_nop1
391             #endif
392             #if (w2_nops&2)
393             w_nop2
394             #endif
395             #if (w2_nops&4)
396             w_nop4
397             #endif
398             #if (w2_nops&8)
399             w_nop8
400             #endif
401

```

```

402     #if (w2_nops&16)
403     w_nop16
404     #endif
405     "        out    %2,%4 \n\t" // Write Low mask (%4) to LED_PORT (%2)
406     #if (w3_nops&1) // w3 nops for timing
407     w_nop1
408     #endif
409     #if (w3_nops&2)
410     w_nop2
411     #endif
412     #if (w3_nops&4)
413     w_nop4
414     #endif
415     #if (w3_nops&8)
416     w_nop8
417     #endif
418     #if (w3_nops&16)
419     w_nop16
420     #endif
421     "        dec    %0    \n\t" // Decrement Loop counter (%0) (Also sets Z
422 if 0x00)
423     "        brne   loop%=\n\t" // Jump to Loop entry point if Z is set.
424     // %0 = 8 bit loop counter
425     : "=d" (ctr)
426     : "r" (curbyte), "I" (_SFR_IO_ADDR(LED_PORT)), "r" (maskhi), "r"
427 (masklo)
428     // %s1 = Data      %2 = LED_PORT      %3 = high mask %4 =
429 low mask
430     );
431 }
432 SREG = sreg_prev; // Restore interrupt status
433 _delay_us(50); // Reset delay
434 }
435 /* ***** END SECTION WS2812 DRIVER ***** */
436
437 void sendLCDNible(volatile uint8_t data, uint8_t rs)
438 {
439     // Mask out fist 4 bits
440     data &= 0b00001111;
441     // Mask in LCD_BACKLIGHT if required (pin 7)s
442     if (LCD_BACKLIGHT) data |= 0b10000000;
443     // Mask in register select
444     if (rs) data |= 0b01000000;
445     // Bit 4 => 1, its the interrput pin, its on pull-up!
446     data |= 0b00010000;
447     // Set Data
448     LCD_PORT = data;
449     // Small delay, data needs to be valid BEFORE enable
450     _delay_us(800);
451     // Toggle enable
452     LCD_PORT |= 0b00100000;
453     // Larger delay, LCD needs time to process
454     _delay_us(800);
455 }
456
457 void sendLCDInstructionByte(uint8_t data)
458 {
459     // rs = 0 -> instruction
460     sendLCDNible(data >> 4, 0);
461     sendLCDNible(data, 0);
462 }
463
464 void sendLCDCharacterByte(char data)
465 {
466     // rs = 1 -> data
467     sendLCDNible(data >> 4, 1);
468     sendLCDNible(data, 1);

```

```
469     }
470
471     void sendLCDBuffer(char * buffer)
472     {
473         // max chars
474         for (uint8_t i = 0; i < MAX_LCD_CHARS; i++)
475         {
476             // End on 0x00
477             if (buffer[i] == 0x00) break;
478             sendLCDCharacterByte(buffer[i]);
479         }
480     }
```